BASEMENT

MASSACHUSETTS
INSTITUTE of TECHNOLOGY
LIBRARIES

HD28
.M414

No 1260-81

Dewey

JAN 24 1983

LIBRARIES

# TOWARDS AN AXIOMATIC APPROACH TO INFORMATION SYSTEMS DEVELOPMENT

John J. Donovan

Steven H. Kim

August 1981

CISR No. 77
Sloan WP No. 1260-81

Center for Information Systems Research
Massachusetts Institute of Technology
Alfred P. Sloan School of Management
50 Memorial Drive
Cambridge, Massachusetts, 02139
617 253-1000

# TOWARDS AN AXIOMATIC APPROACH TO INFORMATION SYSTEMS DEVELOPMENT

John J. Donovan

Steven H. Kim

August 1981

CISR No. 77
Sloan WP No. 1260-81

To be presented at the Second International Conference on Information Systems

Cambridge
December 1981

© 1981 John J. Donovan and Steven H. Kim

Center for Information Systems Research

Sloan School of Management

Massachusetts Institute of Technology

M.I.T. LIBRARIES
JAN 2 8 1966
RECEIVED

## ABSTRACT

This paper advocates an approach (called the axiomatic method) to reduce the costs of constructing an information system. Further, we contrast the applicability of the axiomatic method to the more traditional approach of enumerating alternatives (the algorithmic method) in constructing an information system.

We delineate the steps involved in building an information system, present a set of pilot axioms, and offer some derivative theorems. We then apply these axioms and theorems to each phase (specification, design, implementation and maintenance) of the information system life cycle, and confirm a number of empirical results other information system builders have observed.

0743614

# 1. INTRODUCTION

Our discussion of PRODUCTIVITY pertains to the reduction in time and cost involved in all four phases of building an information system (specification, design, implementation and maintenance). We include in the term INFORMATION SYSTEM any computer-based system used to store, analyze and present information. This includes the spectrum of systems from operational examples such as payroll accounting to decision support systems for strategic planning.

The purpose of this paper is to present a systematic method for whittling down construction time and cost; this is to be achieved by strategically reducing the number of alternatives that a system builder must evaluate when constructing an information system.

We use the term INFORMATION SYSTEM CONSTRUCTION to apply to all four phases of the system life cycle. Likewise the term INFORMATION SYSTEM BUILDER refers to the person(s) in charge of these four phases.

## 1.1 BACKGROUND ON AXIOMATICS

In some areas such as manufacturing, it has been recognized that certain design decisions (e.g. maintaining modularity) always appear to yield superior designs. This observation

suggests the existence of basic natural principles which govern the process. If these principles or axioms can be tracked down and crystallized for information systems, we may establish a scientific approach to design.

AXIOMS are general truths immune to violations or counter-examples. They are to be taken as first principles and are intrinsically unprovable.

THEOREMS may be defined as readily derivable consequences of axioms, while COROLLARIES are readily deducible results of axioms and theorems.

FUNCTIONAL REQUIREMENTS are the minimum set of independent specifications that completely define the tasks. Examples include the capacity of the database, number of daily transactions, degree of multiprogramming, level of read/write security, and extent of backup to allow complete reconstruction.

In addition to functional requirements, constraints are often needed to specify limits on byproducts or side effects. CONSTRAINTS are specifications that define the boundaries within which attributes of these side effects are acceptable. Examples include upper limits on cooling requirements or mean time between failures, or lower limits on the accuracy of numerical solutions.

White and Booth [21], for example, recommend the inclusion of these specifications for software design:

1. Functional specifications. Definition of each data element and the control structure among various tasks.
2. Performance constraints. Specification of each subtask

to operate within its time and space constraints and to allow the overall task to do the same.

HEURISTICS are similar to axioms in that they both offer working guidelines. But a heuristic doesn't carry the weight of an axiom. Example heuristics might be:

1. "If your average accounts receivable is over $1,000, ignore those under $10."
2. "For clarity and readability, keep subroutines under 30 executable source statements." [20]

These are heuristics because they provide rules of thumb with no claim to ALWAYS yielding the best result. In contrast, axioms by definition are always valid. Perhaps the most famous axioms are those of Thermodynamics, such as the First Law: "The total energy of a system is constant." (The two sample heuristics above may be promoted to axioms if industrial or psychological case studies were to reveal that they always produce the most profitable or readable results.)

## 1.2 FRAMEWORK FOR INFORMATION SYSTEMS DEVELOPMENT

The phases involved in constructing an information system may be classified in a variety of ways [14]. In this paper, we view the development phases as:

1. Specification

a. functional requirements

b. constraints

2. Architectural design

a. hardware

b. software

3. Implementation

a. hardware selection

b. code generation

c. testing

4. Maintenance

These phases are not intended to be strictly partitioned. For example, according to one method of software development, the encoding and testing of modules should proceed side by side.

In addition, a development arising in one phase might necessitate backtracking to an earlier one. In illustration, a program error detected in testing may require a regression to the coding or even the software design stage. In fact, this reverse process occurs often enough to prompt investigation: Boehm, McClean and Urfrig [2] have shown that the cost of correcting errors at coding time is about twice that of changing it at the design stage; and finding it at testing time costs about 10 times that during design.

## 2. ALGORITHMIC VERSUS AXIOMATIC APPROACH

## 2.1 ALGORITHMICS

The techniques for evaluating alternative information systems may be subdivided into two basic types, algorithmic and axiomatic. The axiomatic method is based on a set of rules which efficiently identify the global optimum. In contrast, the algorithmic approach is a procedural method for considering alternatives, and may be subdivided further into two categories: exhaustive and rapid search methods.

### 2.1.1 EXHAUSTIVE SEARCH

One algorithmic approach--EXHAUSTIVE SEARCH--enumerates all possible configurations and choices. The drawback with exhaustive search with respect to information system construction lies in the myriad technical choices and the explosive number of functional requirements demanded of new systems. To illustrate, consider the recent design of an information system at Microwave Associates of Burlington, Massachusetts: some design attributes or dimensions considered were the choice of operating system, size of CPU, type of data representation, and selection of programming language. Suppose--in this fancifully small example--that there are only 10 such attributes to consider, with say 5 choices per attribute. The result is 5**10, or over 9 million, design possibilities. It would take more than a few weekends to evaluate them all.

### 2.1.2 RAPID SEARCH

Another algorithmic approach is RAPID SEARCH, in which a set of guidelines constrain the domain of evaluation. One example is branch and bound, which seeks to discard entire branches of inferior alternatives in the design tree.

Another example is stagewise optimization: as each attribute is optimized, the next attribute is evaluated conditionally under the constraint that the preceding attribute choices will prevail.

Consider again the 4 attributes mentioned. The steps for stagewise optimization are:

Step 1. Identify all attributes and all choices within each attribute. Optionally, these attributes might be ranked in order of importance.

Step 2. Select the best choice of operating system. Call it C1. Suppose C1 = VM/370.

Step 3. Select the size of CPU given that C1 holds. Call it C2. Say C2 = 1024K.

Step 4. Select the data representation, given C1 and C2. Say it's C3 = relational data model.

Step 5. Select the programming language given C1, C2, and C3. Suppose C4 = PL/I.

Then the design with {C1,C2,C3,C4} will be the stagewise optimal.

Suppose there are n attributes, with m choices per attribute. Then the stagewise algorithm requires

$$N_{SW} = \sum_{i=1}^{n} m_i$$

evaluations. In contrast, the exhaustive search method requires

$$N_{es} = \prod_{i=1}^{n} m_i$$

evaluations. The efficacy of the stagewise method for a problem with 5 choices in each of 10 dimensions is

$$\frac{N_{es}}{N_{sw}} = \frac{\prod_{i=1}^{n} m_i}{\sum_{i=1}^{n} m_i} = \frac{5^{10}}{5 * 10} \approx 2 * 10^5$$

In the class of rapid search methods, only a few well-specified techniques (e.g. branch and bound) can lay claim to solution algorithms that yield the global optimum. Usually the drawback of rapid search is its lack of guarantee of producing the global optimum: the union of optimized subproblems does not necessarily yield a global optimum unless the components are mutually independent.

## 2.2 AXIOMATICS

The question now arises: Does there exist a set of general principles which always provides rules for eliminating large subsets of design configurations? The AXIOMATIC APPROACH is an attempt to specify those rules.

# 3. PRESENTATION OF AXIOMS

## 3.1 SOME DEFINITIONS

We may define a FEASIBLE system configuration as one that satisfies the functional requirements and constraints.

Then PRODUCTIVITY may be defined in a Pareto-optimal sense [8], with time and monetary costs as attributes or dimensions. Consider two feasible configurations A and B. In this paper, we say that A is more productive than B if both the time and cost required to build A are less than that of B.

For our purposes, INFORMATION may be defined in a variety of ways [9]. The information content involved in a system design might be characterized by the number of bits needed to encode or fully describe the design. When communication between modules is involved, information might be taken as defined by Shannon [17]:

$$I = \sum_i - p_i \log_2 p_i$$

where $p$ is the probability of transmitting the ith message, and the summation occurs over all possible messages.

The concept of ENTROPY may be defined loosely as randomness or disorder. In the sense of information theory or statistical mechanics, entropy may be defined as the negative of information. This relationship will be explored further in the future.

## 3.2 AXIOMS

We propose the following axioms as the set applicable to the construction of information systems:

A1: Productivity increases when information content is minimized.

A2: Entropy increases over time, or at best remains constant.

A3: Productivity increases when the independence of functional requirements is maintained.

A1 calls for a minimization of information content. Intuitively, we expect the cost and complexity of a particular implementation to rise with a rise in the information that must be used to define the system.

A2 refers to the viability of implemented systems. It implies that randomness or disorder in a system tends to increase over time. Eventually the functional modularity of the system deteriorates to the point where a completely new information system must be built afresh. This concept is discussed further in connection with the theorems given below.

A3 calls for a solution which satisfies the functional requirements independently. A global optimum is difficult to attain when a change in one attribute triggers a change in others: this would require the collective optimization of the entire set of interacting components. In contrast, maintaining independence allows for modular optimization in which

optimization of each component may proceed independently of the others. The more independent the components, the better the solution.

These axioms are adopted from those that have been applied to other fields. A2 is a restatement of the Second Law of thermodynamics; A1 and A3 have been applied by Suh, Bell and Gossard [18] to manufacturing systems.

## 4. APPLICATION OF THE AXIOMS

In this section we describe how the axioms give rise to a number of theorems pertaining to various phases of the information system life cycle. We also indicate how the theorems might be proved.

### 4.1 PHASE 1: SPECIFICATION

The first theorem follows from A1, which calls for a minimization of information:

> T1.1 Productivity increases when the number of functional requirements are minimized.

Obviously the information content incorporated in a system specification can only increase with an increase in the number of functional requirements and constraints. The resulting

complexity can then only increase construction cost. This assertion is consistent also with the behavior of manufacturing systems [22].

## 4.2 PHASE 2: ARCHITECTURAL DESIGN

In the following theorems, the terms "module" and "component" may apply either to hardware or software in the architectural design phase:

T2.1 Productivity increases with increased use of standardized or interchangeable modules.

T2.2 A design should incorporate functional requirements in a single module if these requirements can be kept from mutually interacting.

T2.3 If a design exhibits coupled functional requirements, these requirements should be segregated or decoupled.

The first theorem (T2.1) springs from A1, which requires a minimization of information. When a standard module is used, its description or specification is required only once. If the module is needed again, it may be specified simply by referencing the original module.

T2.2 derives from A1, since a reduction in the number of discrete modules minimizes information that would otherwise be needed to specify how all the original modules would interact. However, as A3 requires, the functional requirements must still

be independent of each other; if not, the interdependencies may result in increased overall information requirements. This is the idea behind theorem T2.3.

The use of T2.3 may be illustrated by an example from one of the writers' personal experience. The setting involved the Pan Am reservation system which was experiencing phenomenal growth. The system retrieved data through linear search methods; but the increasing size of the database led to a corresponding increase in retrieval time, which in turn reduced the daily rate of transactions processed. In this case, the functional requirements pertaining to the size of the database and the transaction processing rate had become coupled.

A decoupling of these functional requirements was in order; this was accomplished by converting the method to hashing [5]. For low record densities in a hashed system, the accessing rate (hence the transaction processing rate) is largely independent of the database size.

### 4.2.1 PARTITIONING OF FUNCTIONAL REQUIREMENTS

The complete independence of functional requirements is an ideal to strive for, but may be difficult to attain in practice. As a practical matter, a partial independence among subsets of functional requirements may be better than none.

We may invoke A1 and A2 to yield the following theorem:

T2.4 Functional requirements should be partitioned into

smaller groups with minimal interaction between groups.

Consider a financial applications package, for example. According to this theorem, a change in the credit check module should not disturb the billing module.

To optimally partition the functional requirements into smaller groups, it is important to first evaluate the relationships between those requirements. These functional dependencies may then be represented by an undirected graph (See, for example, Andreu [1] and Huff [7]). Wong [23] presents a brief survey of existing graph-decomposition techniques and offers a better method for finding subgroups of independent functional requirements. His technique is discussed in greater detail in the Attachment.

## 4.3 PHASE 3: IMPLEMENTATION

From axiom A1 we also propose another theorem applicable to Phase 3b (software design) of the information systems development life cycle:

T3.1 The number of program statements should be minimized.

This is consistent with the empirical observation that cost increases disproportionately with program size [11]:

$$\text{Effort} = \text{Constant} * (\text{Number of instructions})^{1.5}$$

Another theorem due to A1 is

T3.2 Productivity increases with the use of a higher-level language.

Taliaffero [19] reports that productivity is constant at 2,400 statements per year whether a program is written in assembler, Fortran or Cobol. Nelson [12] also shows an increase in productivity by a factor of 3 or more by using higher-level languages.

Some other consequences of A1 are:

T3.3 A system should be decomposed into smaller logical units.

T3.4 Separate subroutines should be designed for each elementary task.

Here, information is minimized because the interaction among the elements of the system are localized within each module. In this situation, any interaction between modules i and j are accomplished as components in their entirety, without the need for one to keep track of the function of each element within the other module.

One rule of thumb calls for a partitioning of functions into modules until each module includes no two elements that might be useful in isolation. Another rule claims that each program

module should be small enough to fit on one page, thereby allowing comprehension at a single glance.

The drive toward modularity is not costless, of course. Camp and Jensen [4] report that modularity results in an extra 20-35% overhead in memory and another 10-15% excess in run time over a monolithic architecture; but these costs are small in comparison with the major savings in productivity during development and maintenance. In fact, complexity and costs triple as module size doubles. This may be compared with the concensus opinion of software analysts, who believe that doubling the module size will increase complexity and cost by 50 to 100%. As a result, the recommended average module size is 3 to 5 times the average module overhead. For example, if the module overhead is 10 words, then the average module size should be 30 to 50 words, including the overhead.

A1 and T3.1 suggest this result:

T3.5 The number of instructions coded is not a measure of productivity.

Intuitively, large-scale systems will require many program instructions. But A1 and T3.1 call for a reduction in the number of instructions generated. Hence productivity cannot be measured by total program size.

## 4.4 PHASE 4: MAINTENANCE

Axiom A2 suggests the following theorem:

T4.1 The usability of a system decreases over time, and will eventually vanish.

Brooks [3] maintains that all information systems die eventually. This is attributed to the inevitable patches or fixes to software errors, and the resulting decay in the conceptual integrity of the sytem. The need for fixes arise from a variety of factors.

a) Bugs.

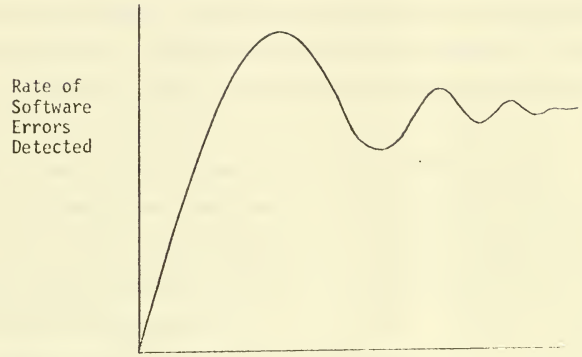
Bugs in large systems are remarkably hardy creatures. According to Brooks [3], fixing one error will merely introduce another with 20 - 50 % probability.

Pikul and Wojcik [15] offer the following model for verminous attacks. As shown in the diagram, the rate of bugs detected rises steeply at the outset of any program. As debugging proceeds in earnest, the attack rate eventually peaks, then drops. But it rises once again, to oscillate around a steady-state value within an attenuated envelope.

A highly damped version of this model (steep rise and slow decay to steady-state value, without the minor oscillations) is supported also by Ramamoorthy and Ho [16].

b) Changes in user requirements.

In practice, as users become proficient with a particular system, they begin to demand higher performance. They change the original functional requirements by stretching an existing one or even even adding new ones. An airline




reservation system, for example, may begin operation as a simple passenger booking system. In due course the system is expanded to allow for kosher meals, flight scheduling, fuel distribution, and other functions. The rate of addition of new code could easily mean that the bugs are proliferating faster than they are being eliminated.

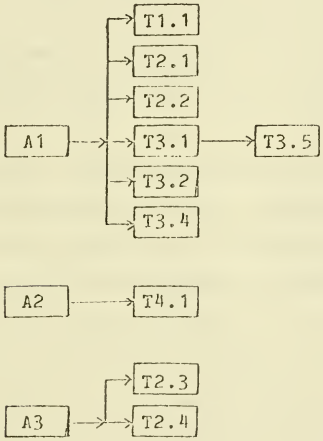
c) Changes due to hardware.

Technological advances may dictate the switch from, say, an IBM/370 to a /3033 for increased processing speed. Any

such transformation is rife with conversion problems.

## 5. RELATIONSHIPS BETWEEN THEOREMS

The precedence relationships among the axioms and theorems may be summarized as follows, where arrows indicate the relationship "derived from":




## 6. RELATIONSHIP TO CREATIVITY

A question which might arise is, "What role does creativity play in building information systems?" Creativity precedes analysis in the construction of information systems; it is important in the generation of alternative designs, without which there can be no comparative evaluation.

A variety of methods are available for stimulating creativity [6]. One of these is the trigger word method involving questions about what the design is supposed to do. The checklist method [13] is based on a series of questions on modification, such as "Magnify?", "Rearrange?" or "Combine?"

The morphological method [24] requires determining the attributes involved, listing them, and considering all the resulting combinations.

Brainstorming refers to the animated generation of ideas by a heterogeneous group of participants, some of whom may be entirely new to the concepts under discussion. The purpose is to produce as many ideas as possible, however unorthodox they may be.

This paper, however, is not intended to address creativity per se. The aim of axiomatics is to channel creativity by providing a set of guidelines. The objective lies not in the generation of designs, but in the assignment of relative merit to alternative configurations.

## 7. CLOSURE

This paper has introduced the application of the axiomatic approach to productivity in constructing information systems. We have enumerated three pilot axioms, proposed a number of theorems, and indicated how the theorems spring from the axioms and draw upon empirical results from the construction of information systems.

We note that axiomatics is not a methodology for generating candidate designs, but a tool for use in the decision-making process of evaluating them. Further, axiomatics is not intended to supplant algorithmics. Rather, they will reinforce each other in streamlining the development process, as illustrated by the use of the decomposition algorithms discussed in Section 4.2.1.

We hope that this paper will start a dialogue within the informations systems community, so that we may collectively generate the analytical and experimental results needed to carry this concept further. The axioms and theorems proposed here are only a pilot set, and may require changing, deleting or adding to in the light of experience.

In the future we would like to refine the axioms into a compact set, to rephrase them in a more quantitative form from which to prove the theorems, and to validate them through case studies in systems development. The successful development of the axiomatic approach should open up new avenues for increasing productivity in the construction of information systems.

ACKNOWLEDGEMENTS

We gratefully acknowledge our debt to those who have written before us, both in information systems and in the application of the axiomatic method in other fields. Recognition is due in particular to Professor Nam P. Suh and his colleagues at the MIT Laboratory for Manufacturing and Productivity, and to those in the field of thermodynamics who initially applied the axiomatic method to the analysis of physical systems. We also thank our colleagues at the Sloan School of Management-- Tony Wong, Jim Lattin and Mike Treacy--for their thoughtful comments and suggestions.

## ATTACHMENT: DECOMPOSITION OF FUNCTIONAL REQUIREMENTS BY THE HIGH-DENSITY CLUSTERING METHOD

### A.1 GRAPHICAL REPRESENTATION OF FUNCTIONAL REQUIREMENTS

The first task in partitioning functional requirements is to represent them as nodes of a graph, and the interdependencies between them as arc weights. In building an information system, some examples of functional requirements might be:

1. The users will be guided by menus.
2. A report-writing facility will allow users to develop customized reports.
3. Reports can be directed to the line printer or the user's terminal.

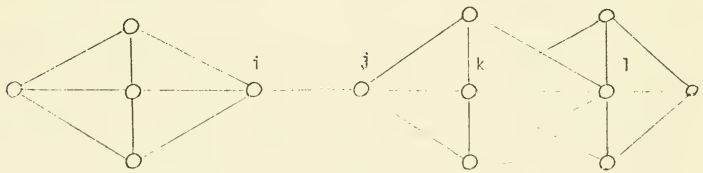
Each of these functional requirements may be represented graphically as a node in a design graph. For each pair of functional requirements, we can consider the degree of interrelationship between the nodes. The extent of this association can be characterized as the weight on the link or arc between the pair.

For convenience the weights are normalized between 0 and 1. If two functional requirements are deemed to have a weak interdependency, the system builder might assign a link weight of 0.3; an average degree of coupling may be represented by 0.5; a strong relationship by 0.8. If two functional requirements are deemed independent, the link weight is 0.0, and the link itself is eliminated from the design graph.

Returning to our example above, the first and second functional requirements may be considered independent and therefore assigned a link weight of 0 (i.e. no link between the two nodes). But the first and third may be viewed to have an average degree of interdependency, since the report directing aids must be made available to the user. So the link weight is given a value of 0.5. In this way the set of functional requirements and their interdependencies may be represented graphically.

## A.2 DENSITY CONTOURS

This section outlines the high-density clustering method for functional decomposition proposed by Wong [23]. Consider a graph consisting of nodes and unweighted arcs. Intuitively, two nodes belong to the same group or cluster if they are linked to each other and to many nodes in common. In the figure below, nodes k and l should belong in the same cluster, while nodes i and j



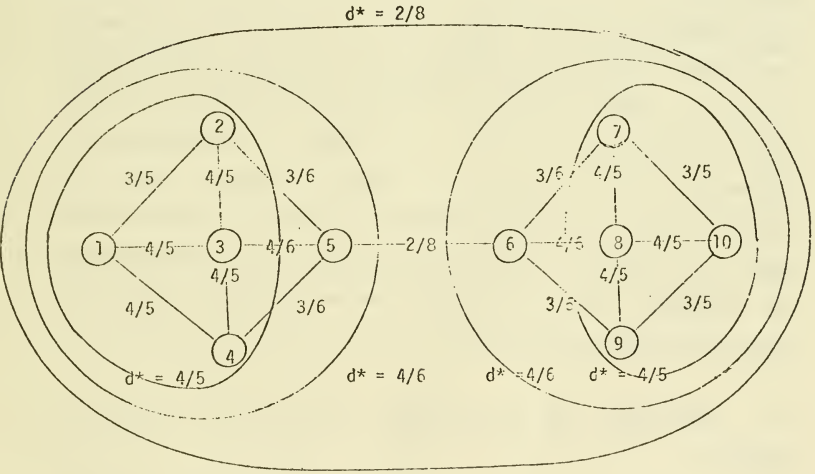

should be in separate clusters. The DENSITY CONTOUR on the link

between any two nodes i and j is defined as:

$$d_{ij} = \frac{\bigcap N_{ij}}{\bigcup N_{ij}} = \frac{\text{No. of nodes connected to both i and j (including i and j}}{\text{No. of nodes connected to either i or j or both (includi i and}}$$

If two nodes are unlinked, their contour is defined to be zero.

The figure below illustrates the use of this definition. For example, the contour between nodes 1 and 2 is 3/5.




For weighted arcs, the corresponding definition for the density

contour is

$$d_{ij} = \frac{2\,W_{ij} + 1/2 \sum_{k \in C} (W_{ik} + W_{jk})}{\bigcup N_{ij}}$$

where $W_{ij}$ = weight on the link between nodes i and j; C = set of nodes connected to both i and j (excluding i and j).

Now we turn to the idea of grouping individual nodes. A HIGH DENSITY CLUSTER at level d* on a graph G, is a subgraph S such that S is maximal among connected sets of nodes whose nodes are connected by links with density contour $\geq$ d*. The nested loops in the preceding figure represent the density contours for the given graph.

The family of high-density clusters on a graph may be shown to form a tree. As the density level d* is decreased, the cluster S at level d* expands smoothly. This gradual expansion occurs until a splitting level $d_S^*$ is reached, at which point the cluster joins with a previously disjoint cluster. These two clusters, called BRANCHING CLUSTERS, are useful in suggesting the number of subgraphs in the original graph. In the diagram above, the splitting level is $d_S^* = 2/8$; the two branching clusters are {1,2,3,4,5} and {6,7,8,9,10}.

## A.3 IDENTIFYING AND PARTITIONING THE TREE OF HIGH-DENSITY CLUSTERS

Consider a set of N nodes with density contours d(i,j). The

algorithm to identify the tree of high-density clusters is:

STEP1. Let i and j be the pair of nodes with densest link. Combine them to form a cluster l; define the density contour between that cluster and any node k by

$$d(l,k) = \max [d(i,k),d(j,k)]$$

STEP2. Repeat STEP1, treating l as a node and ignoring i and j. The aggregation of nodes continues until all nodes are absorbed into a single large cluster.

The foregoing procedure is equivalent to the minimum spanning tree algorithm. The drawback of this procedure is that the tree of clusters does not explicitly yield the optimal grouping of functional requirements. This last step may be effected by an algorithm given by Lattin [10], which identifies the optimal subgraphs of functional requirements from the tree.

REFERENCES

[1] Andreu, R.C. A systematic approach to the design and structuring of complex software systems. Unpublished Ph.D. thesis, Sloan School of Management, MIT, 1978.

[2] Boehm, B., McClean, R. and Urfrig, D. Some experience with automated aids to the design of large scale reliable software. Proc. International Conference on Reliable Software, Los Angeles, CA, April 1975, pp. 105-113.

[3] Brooks, F.P., Jr. The Mythical Man-Month: Essays on Software Engineering. Addison-Wesley, Reading, MA, 1975.

[4] Camp, J.W., and Jensen, E.P. Cost of modularity. Proc. Symposium on Computer Software Engineering, April 20-22, 1976, Polytechnic Press, New York, 1976.

[5] Donovan, J.J. Systems Programming. McGraw-Hill, New York, 1972, pp. 91-95.

[6] Harrisburger, L. Engineersmanship. Brooks/Cole Publising, Belmont,CA, 1966.

[7] Huff, S.L. A systematic methodology for designing the architecture of complex software systems. Technical Report No. 12, Naval Electronic Systems Command, Washington, D.C., 1979.

[8] Keeney, R.L. and Raiffa, H. Decisions With Multiple Objectives. John Wiley, New York, 1976, pp.69-77.

[9] Kim, S.H. Manufacturing axiomatics, with special application to air compressor design. Unpublished S.M. thesis, Department of Mechanical Engineering, MIT, 1978.

[10] Lattin, J.M. Implementation and evaluation of a graph partitioning technique based on a high-density clustering model. Technical Report #15, Center for Information Systems Research, MIT, 1981.

[11] Nanus, B. and Farr, L. Some cost contributors to large-scale programs, AFIPS Proc. SJCC, Spring 1964, pp. 239-248.

[12] Nelson, E.A. Management Handbook for the Estimation of Computer Programming Costs. Systems Development Corp., Report TM-3225, pp. 66-67.

[13] Osborn, A.F. Applied Imagination. Charles Scribner's Sons, New York, 1963.

[14] Peters, L.J. and Tripp, L.L. A model of software engineering. Proc. 3rd International Conference on Software Engineering, Atlanta, GA, May 10-12, 1978, pp. 63-70.

[15] Pikul, R.A. and Wojcik, R.T. Software effectiveness: a reliability growth approach. Proc. Symposium on Computer Software Engineering, April 20-22, 1976, Polytechnic Press, New York, 1976.

[16] Ramamoorthy, C.V. and Ho, B.F. Testing large software with automated software evaluation systems. IEEE Transactions on Software Enginerring, SE-1, No. 1, pp. 46-58.

[17] Shannon, C.E. The Mathematical Theory of Communication. Univ. of Illinois Press, Urbana, IL, 1949.

[18] Suh, N.P., Bell, A.C., and Gossard, D.G. On an axiomatic approach to manufacturing and manufacturing systems. Trans. of the ASME 77-WA/PROD- 14, 1977.

[19] Taliafero, W.M. Modularity, the key to system growth potential. Software, 1, 3 (July 1971), pp. 245-57.

[20] Weinberg, G.M. PL/I Programming: A Manual of Style. McGraw-Hill, New York, 1970.

[21] White, J.R. and Booth, T.L. Towards an engineering approach to software design. Proc. 2nd International Conference on Software Engineering, Oct. 13-15, IEEE, 1976, pp. 214-222.

[22] Wilson, D.R., Bell, A.C., Suh, N.P., van Dyck, F., and Tice, W.W. Manufacturing axioms and their corollaries. Proc. North American Metalworking Research Conference, May 13-16, 1979.

[23] Wong, M.A. A graph decomposition technique based on a high-density clustering model on graphs. Technical Report #14, Center for Information Systems Research, MIT, 1980.

[24] Zwicky, F. Discovery, Invention, Research through the Morphological Approach. MacMillan, New York, 1969.

00 5 '82

BASEMENT
Date Due

Lib-26-67

HD28.M414 no.1260- 81
Donovan, John /Towards an axiomatic ap
743614 D*BKS 00136569
3 9080 002 042 759